ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

SISTEMA DETECTOR Y CONTADOR DE TABLONES DE MADERA

Aplicación Android para la identificación, localización y conteo de tablones de madera

DETECTOR AND COUNTER SYSTEM OF WOODEN PLANKS

Android application for identification, location and count of wooden planks

Realizado por
Manuel Ruiz Moncayo
Tutorizado por
Enrique Domínguez Merino
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO 2017

Fecha defensa:
El Secretario del Tribunal

Resumen

Este Trabajo de Fin de Grado se ha desarrollado con el objetivo de realizar un conteo, localización y detección de un conjunto de tablones de madera capturados en una imagen, haciendo uso de un dispositivo Android. Para su realización se han utilizado diversas técnicas de visión por computador, centrandose principalmente en el algoritmo detector de bordes Canny y la transformada de Hough, con la cual, se obtienen la líneas delimitadoras del tablón de madera.

El proyecto está basado en el trabajo de los demás integrantes del equipo, encargados de la investigación e implementación en Matlab del algoritmo detector de tablones.

Todo el algoritmo ha sido implementado en C++, debido a la eficiencia de tiempo y recursos que ofrece la implementación en este lenguaje, y luego ha sido enlazado con Android utilizando el *framework* NDK. Para su implementación en C++ y Android se ha hecho uso de un conjunto de herramientas ofrecidas por OpenCV, pues facilita el desarrollo de algoritmos de procesamiento de imágenes.

Palabras claves

Android, visión, computador, detector, procesamiento, imagenes, videos, canny, hough.

Abstract

This Final Year Project has been developed with the purpose of counting, locating and detecting a set of wooden planks captured in an image, making use of an Android device. Various techniques of computer vision have been used, focusing mainly on the Canny edge detector algorithm and the Hough transform, with which the boundary lines of the wooden plank are obtained.

The project is based on the work of the other members of the team, in charge of the research and implementation in Matlab of the plank detector algorithm.

The algorithm has been implemented in C ++ because of the time and resource efficiency offered with the implementation in this language, and has been linked to Android using the NDK framework. For its implementation in C ++ and Android has made use of a set of tools offered by OpenCV,so it facilitates the development of algorithms of image processing.

Key words

Android, vision, computer, detector, processing, images, video, canny, hough.

Índice general

Índice de figuras

Índice de Códigos

1. Introducción

1.1. Motivación

Actualmente las empresas dedicadas al tratamiento o transporte de maderas realizan de forma manual el control del número de tablones existente en una gran cantidad de paneles, siendo un operario quien realiza este procedimiento, con el riesgo de error que conlleva. Esto puede ocasionar grandes pérdidas a la empresa debido al descontento de los clientes y a la pérdida de tiempo que supone realizar esta operación de forma manual, así como la comprobación de que no ha habido ningún error.

Por esta razón, se ha decidido realizar una herramienta móvil que posibilite la automatización de este procedimiento, lo que conllevaría a una disminución del tiempo respecto al proceso manual y una minoración de las pérdidas económicas asociadas.

La implementación de las librerías necesarias para la herramienta móvil se basará en la investigación y estudio realizado por los el resto de integrantes del grupo que componen este trabajo de fin de grado.

1.2. Objetivos

En este proyecto se desarrolla una aplicación móvil, implementada en Android para la automatización el proceso de control del número de tablones de maderas pertenecientes a una misma imagen. Este algoritmo estará centrado en la realización de una buena detección, pero atendiendo principalmente al tiempo y al consumo de memoria *RAM*.

La obtención de la imagen de entrada tendrá la opción de ser capturada desde la propia aplicación o importada desde el sistema de archivos del dispositivo.

La aplicación generará una imagen de salida, la cual podrá ser guardada en el sistema de archivos o compartida con otras aplicaciones. Esta imagen de salida contendrá una serie de rectángulos superpuestos en la imagen de entrada que indicarán la posición de los tablones detectados, así como su número.

1.3. Metodología

La metodología seguida ha sido la siguiente:

- Se han realizado pequeñas planificaciones diarias.
- Se ha utilizado la herramienta web *Trello* para simular las pizarras y *poss-its*.
- Se han realizado *feedbacks* continuos para comprobar el correcto funcionamiento de la aplicación y corrección de errores.
- Se han marcado distintas fechas de finalización para cada fase y entregas internas de prototipos.
- Se ha realizado una planificación antes del inicio de cada fase para marcar los hitos y tareas a realizar.
- Se han realizado distintas reuniones con el tutor para supervisar el progreso de la aplicación.

1.4. Entorno Tecnológico

1.4.1. C++

C++ es un lenguaje de programación inventado con el objetivo de extender y agilizar la programación en C. Es denominado un lenguaje multi-paradigma, pues después de su desarrollo en 1980 se le añadieron facilidades para la programación imperativa y programación orientada a objetos.

Para el desarrollo de este proyecto se ha implementado una librería desarrollada en este lenguaje, puesto que ofrece la flexibilidad y eficiencia necesaria para ejecutarlo en una aplicación móvil, que es uno de los puntos más importantes de la aplicación.

1.4.2. OpenCV

OpenCV es una librería que ofrece un conjunto muy amplio de funcionalidades dedicadas al tratamiento de imágenes e implementada en distintos lenguajes, entre ellos: Java, C, C++, Python, etc. Ésta dispone de una comunidad muy amplia de más de 47000 desarrolladores, permitiendo así un proceso de aprendizaje bastante rápido y es distribuida bajo una licencia de software libre, concretamente, la licencia BSD.

En el desarrollo del proyecto ha sido utiliza para la implementación del conjunto de funcionalidades para la detección de tablones, utilizando su implementación en C++ y Java. Ésta ha sido utilizada debido al conjunto de funcionalidades ofrecidas para el desarrollo, eficiencia,

facilidad de uso, documentación detallada y con uso de ejemplos y por el soporte ofrecido para dispositivos Android.

1.4.3. Java

Java es un lenguaje de programación creado con la intención de permitir ejecutar los programas desarrollados en éste en distintas plataformas sin tener que ser recompilados. Es un lenguaje orientado a objetos, y permite el desarrollo de programas concurrentes.

Actualmente, muchas universidades y otros centros de enseñanza utilizan este lenguaje como base del aprendizaje del paradigma orientado a objetos, convirtiéndose en uno de los lenguajes más usados tanto a nivel académico como a nivel empresarial, pues cuenta con más de 9 millones de desarrolladores por todo el mundo. Este lenguaje es, además, utilizado para el desarrollo de aplicaciones de escritorio, aplicaciones web y aplicaciones de dispositivos móviles, principalmente Android, aunque también puede ser utilizado para aplicaciones de dispositivos embebidos.

1.4.4. Android

Android es un sistema operativo desarrollado por Google, basado en el núcleo Linux. Éste fue desarrollado, principalmente como sistema operativo para *smartphones* y *tablets*, pero actualmente, su uso se ha extendido a muchos más dispositivos como televisores, relojes y automóviles. Dispone de una amplia comunidad de usuarios y desarrolladores con más de un millón de aplicaciones registradas en su tienda oficial *Play Store*.

El lenguaje de programación utilizado para el desarrollo de aplicaciones es fundamentalmente Java para la implementación del modelo y del controlador, y XML para el desarrollo de la interfaz gráfica, aunque permite la integración con otros lenguajes mediante el uso de una serie de herramientas.

Se ha decidido utilizar este sistema operativo debido a la experiencia en el desarrollo de sistemas utilizando Java, al anterior uso de este en otros proyectos y a la posibilidad de usar código nativo desarrollado en C++ e integrarlo en un dispositivo móvil.

1.4.5. Android NDK

Android *Native Development Kit* es un conjunto de herramientas que permiten integrar librerías implementadas en C o C++ en una aplicación Android haciendo uso de una clase que actúa como interfaz entre el código nativo y Java.

[0]https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/OpenCV_Logo_with_text_svg_version.svg/1200px-OpenCV_Logo_with_text_svg_version.svg.png

Este conjunto de herramientas agrega una complejidad adicional al proceso de desarrollo, haciendo que su uso no sea adecuado para varios tipos de aplicaciones que no necesitan una gran exigencia computacional, pues, Android NDK está principalmente orientada a procesos muy costosos computacionalmente y que no sean llamados de forma continuada, ya que la llamada desde la interfaz Java al código C o C++ es muy costosa.

1.4.6. MagicDraw

MagicDraw es una herramienta software de modelado y análisis de aplicaciones creado por *No Magic, Inc*, basada en el estándar UML. Es considerada una herramienta CASE (*Computer Aided Software Engineering*), ya que está orientada a facilitar el diseño y mantenimiento de proyectos software, especialmente para diseños orientados a objetos. Permite la creación de diferentes diagramas ofreciendo así una vista global del diseño y funcionamiento del sistema software.

Esta herramienta ha sido utiliza debido a la experiencia de uso obtenido en otros proyectos anteriores y a la licencia ofrecida por la Universidad de Málaga.

2. Especificación y diseño

2.1. Requisitos

2.1.1. Requisitos funcionales

Con el objetivo de mostrar las diferentes acciones disponible para un usuario del sistema y la respuesta de la aplicación, se ha realizado una lista de dichas acciones definidas en los siguientes requisitos funcionales del proyecto:

- El usuario podrá seleccionar una imagen de la galería del dispositivo.
- El usuario podrá realizar una imagen utilizando la cámara del dispositivo.
- El usuario podrá aceptar la imagen cargada para su posterior análisis.
- El usuario podrá realizar un recorte, en el cual, el sistema muestra un rectángulo con el área seleccionada para dicho recorte.
- El usuario podrá cancelar la realización del recorte y se mostrará la imagen original.
- El usuario podrá reintentar la realización del recorte, el sistema mostrará la imagen original para volver a realizarlo.
- El usuario podrá cancelar la opción de seguir a la pantalla de selección del tamaño de rectángulo más común, el sistema volverá a la pantalla de inicio de la aplicación.
- El sistema le mostrará al usuario una interfaz similar al recorte para seleccionar una ventana de un tamaño que se adecue a la mayoría de tablones.
- El usuario podrá aceptar el rectángulo seleccionado para su posterior análisis de la imagen completa.
- El usuario podrá cancelar la selección del rectángulo, el sistema volverá a la pantalla de realización de recorte.
- El sistema analizará la imagen seleccionada haciendo uso del tamaño del rectángulo, mostrará en el proceso una ventana de carga.
- El sistema una vez terminada la detección de tablones mostrará la imagen con los tablones detectados así como su número.

- El usuario tendrá la opción de guardar en el dispositivo la imagen con los tablones detectados.
- El usuario tendrá la opción de cancelar una vez el algoritmo de detección de tablones a terminado, el sistema volverá a la pantalla de inicio de la aplicación.
- El usuario tendrá la opción de compartir una vez el algoritmo de detección de tablones a terminado.

2.1.2. Requisitos no funcionales

Se debe hacer hincapié en un conjunto de restricciones de la aplicación desarrollada, debido a su implementación para dispositivos móviles y a las funcionalidades realizadas. Este conjunto de restricciones son definidas en la siguiente lista de requisitos no funcionales:

- La imagen elegida por el usuario debe ser de un tamaño menor que la memoria *RAM* máxima facilitada por el sistema operativo, pues cambia dependiendo de las características del mismo, si esta es mayor la aplicación se cerrará y mostrará un mensaje.
- La versión mínima soportada del sistema será la versión Android 4.1 (*API* 16).
- El tiempo de ejecución del algoritmo de detección de tablones debe ser óptimo, teniendo en cuenta que es ejecutado en un dispositivo móvil donde la capacidad de procesamiento y memoria *RAM* es generalmente muy limitada.

2.2. Diseño

2.2.1. Fases del proyecto

El proyecto, en general, se puede dividir en distintas etapas realizadas para su desarrollo. Estas etapas no han sido seguidas en el orden expuesto, debido a los distintos *feedbacks* realizados en los distintos hitos marcados a lo largo del proyecto, especialmente la etapa de pruebas que ha sido realizada en cada iteración y para cada función realizada.

- **Análisis de Requisitos:** se han analizado las funcionalidades requeridas por el sistema, generando un conjunto de requisitos.
- **Búsqueda de información:** se ha realizado diferentes tarea de investigación de las diferentes tecnologías utilizadas, así como los distintos algoritmos de procesamiento imágenes para mejorar el tiempo de ejecución y detección para los distintos entornos de la imagen en el algoritmo desarrollado por los demás integrantes del proyecto.
- **Preparación del entorno de trabajo:** se han descargado e instalado las herramientas software necesarias.

- **Diseño de la aplicación:** se han diseñado y modelado los distintos diagramas UML necesarios para la comprensión del sistema.
- **Diseño de la interfaz:** se ha diseñado la interfaz del sistema y la integración entre vistas.
- **Desarrollo de la librería C++:** se ha implementado un conjunto de funciones necesarias para el procesamiento de las imágenes en Android y la detección de tablones.
- **Desarrollo aplicación Android:** se ha implementado la aplicación Android.
- **Fase de pruebas:** se ha probado que los resultados de cada prototipo son los esperados.

2.2.2. Casos de uso

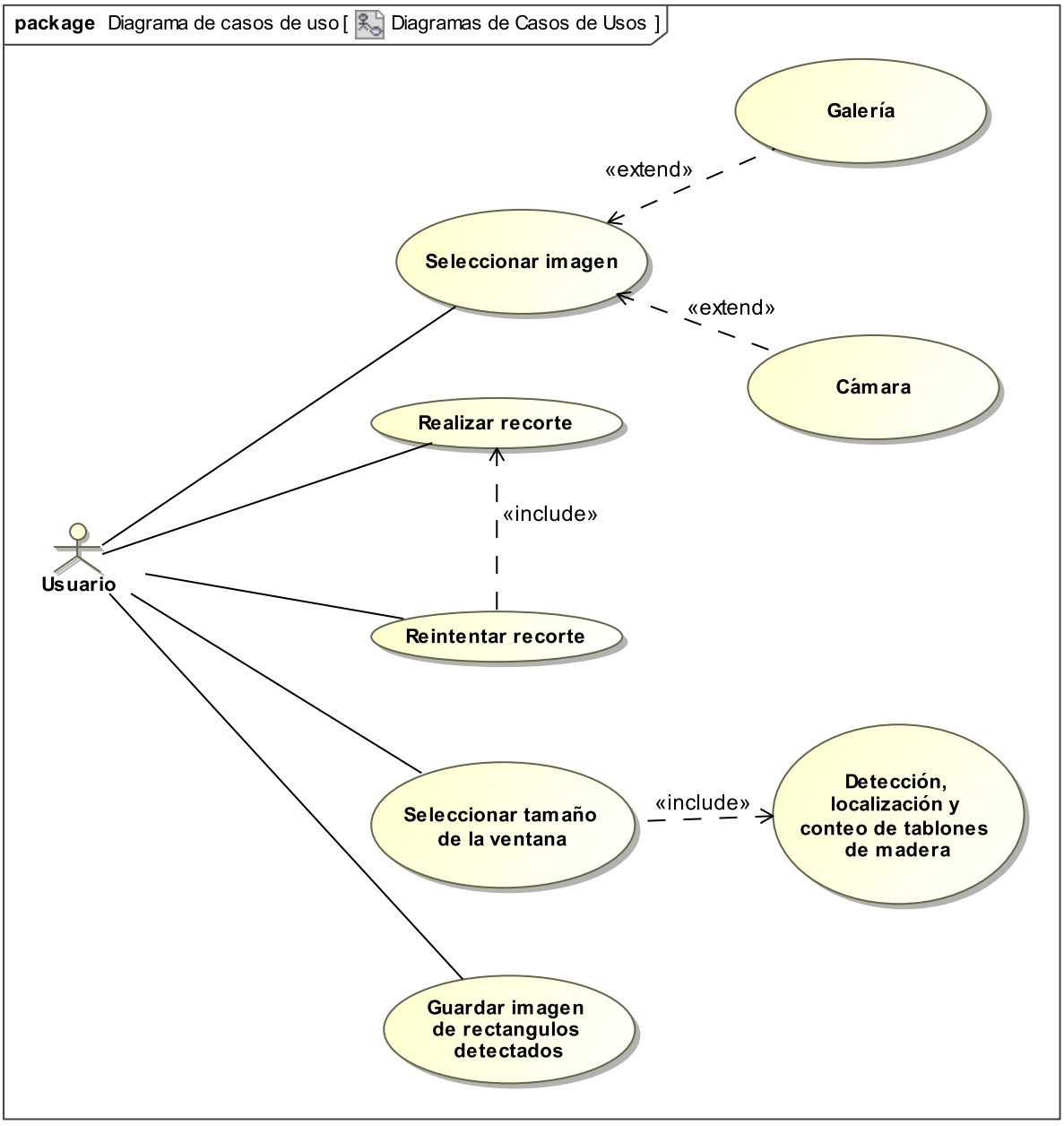


Figura 2.1: Diagrama de casos de uso

2.2.3. Diagrama de clases

C++ y Java

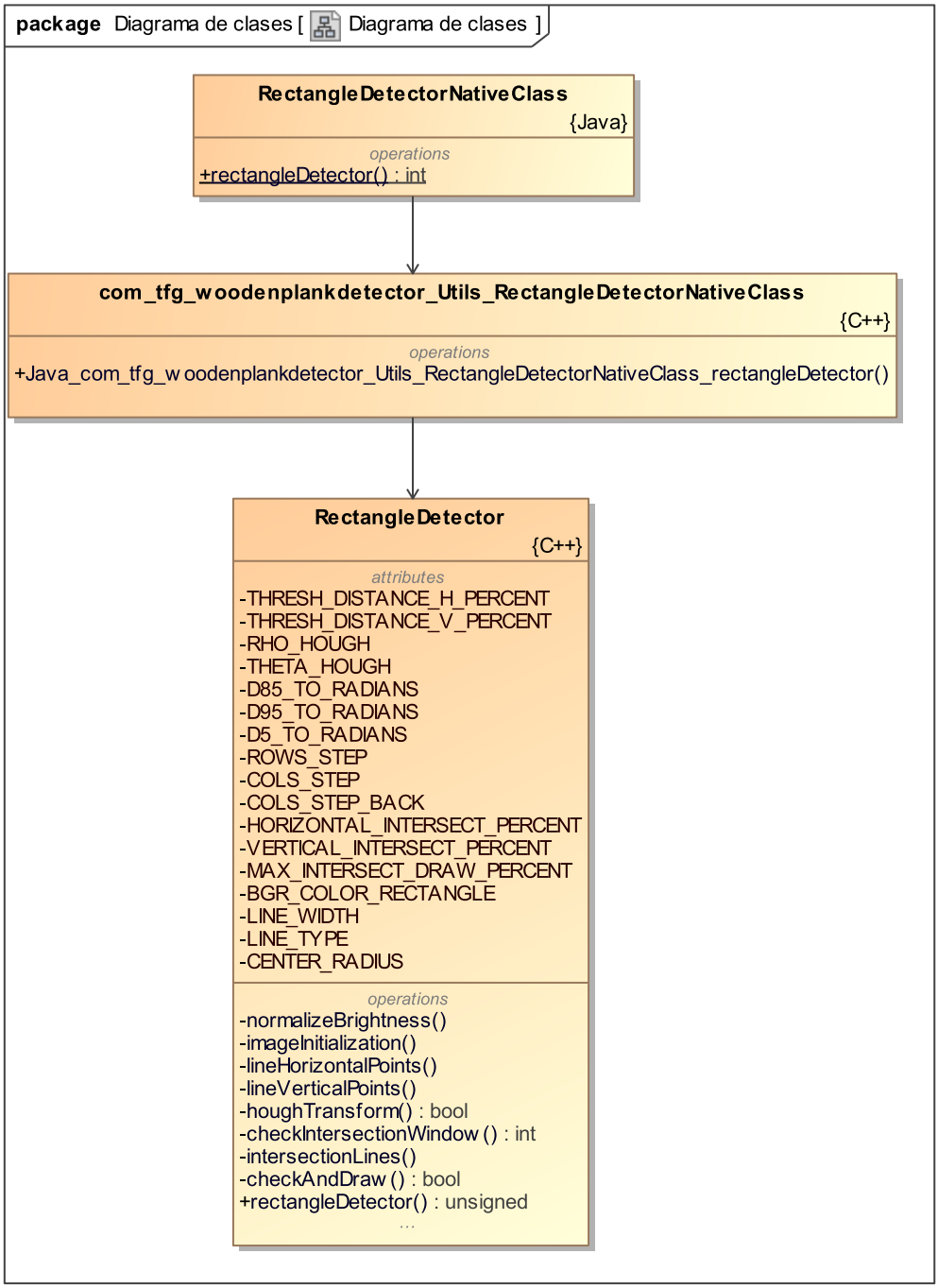


Figura 2.2: Diagrama de clases de la interacción C++ y Java

2.2.4. Diagrama de Secuencia

Librería C++

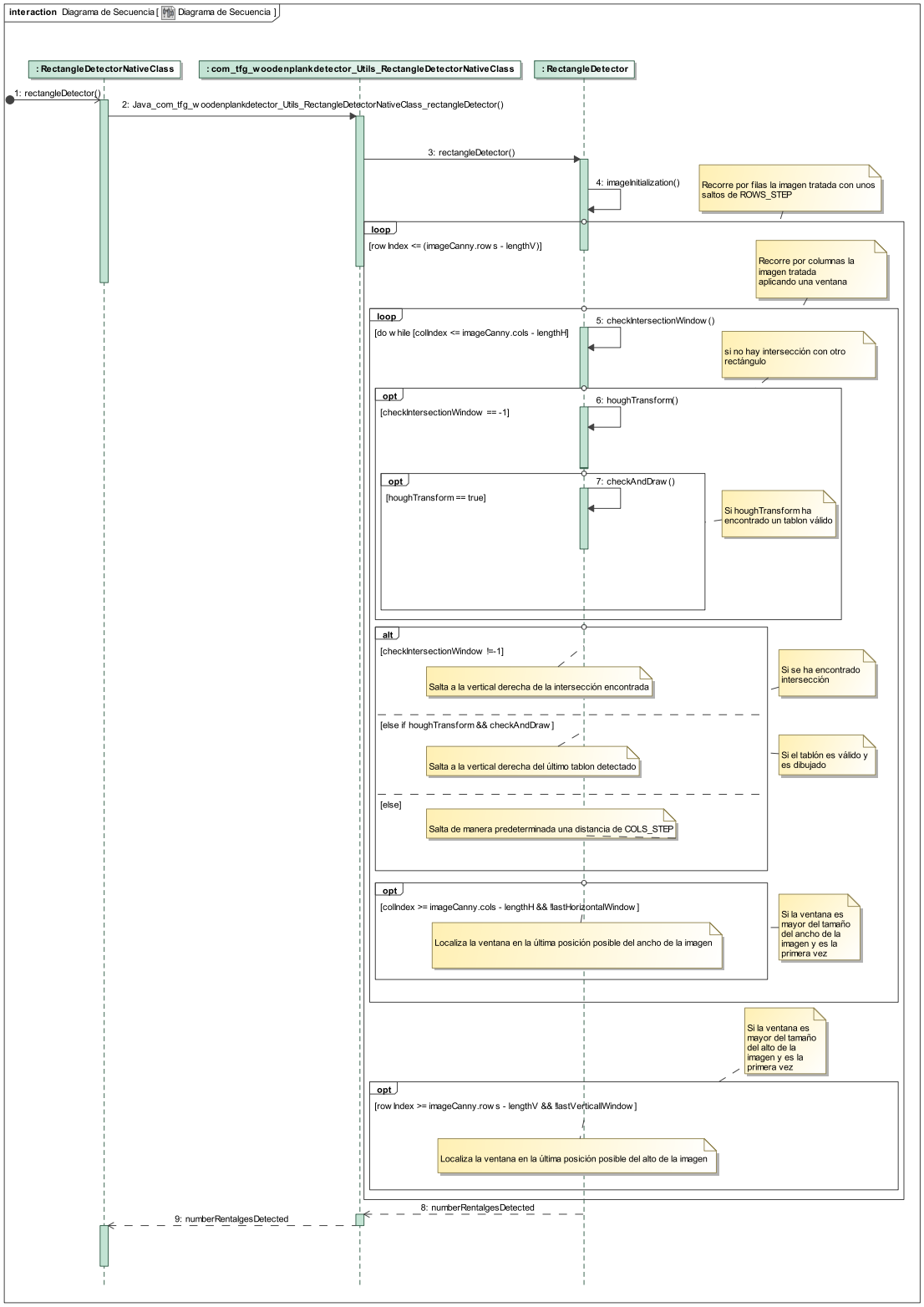


Figura 2.3: Diagrama de secuencia de la función principal de la librería C++

3. Investigación y desarrollo

Durante el desarrollo del proyecto se utilizará los términos rectángulo y tablones de madera de forma indistintiva.

3.1. Librería C++

El desarrollo de la librería C++ para el tratamiento de las imágenes y detección de tablones ha sido implementada en Eclipse, un IDE (*Integrated Development Environment*) que nos ofrece un conjunto de herramientas útiles para la implementación de la librería, facilita la detección de errores y el desarrollo de pruebas con mayor rapidez que en el IDE utilizado para el desarrollo Android.

3.1.1. Instalación de la librería OpenCV

Para comenzar con el desarrollo se ha realizado la instalación de la librería OpenCV para C++ utilizando el ejecutable ofrecido por la página oficial, compilandola utilizando *CMake* y realizando una serie de configuraciones para su integración con Eclipse. Entre ellas la especificación del directorio de localización de la librería y el conjunto de librerías que podrían ser usadas en el proyecto de las ofrecidas por OpenCV. (Figura 3.1)

Inicialmente, todo el conjunto de librerías ofrecidas por OpenCV han sido insertadas en el IDE, pues al ser el entorno de desarrollo de la librería C++ no afecta a su posterior uso en Android, donde se realiza una nueva instalación de OpenCV diferente a la utilizada en Eclipse.

3.1.2. Desarrollo de la librería

El algoritmo de detección de tablones realizado, se basa principalmente en el uso de una ventana deslizante que es desplazada a lo largo de toda la imagen recibida para el tratamiento. Esta ventana debe tener un tamaño suficiente para que de manera intrinseca, intersecte a la mayoría de los tablones permitiendo su estudio mediante la división en porciones de la imagen, en las que se aplica la transformada de Hough para obtener un conjunto de lineas. Este conjunto de lineas son sometidas a un filtrado suficiente para determinar si pertenecen al rectángulo que queda contenido en la ventana en la que se realiza el estudio.

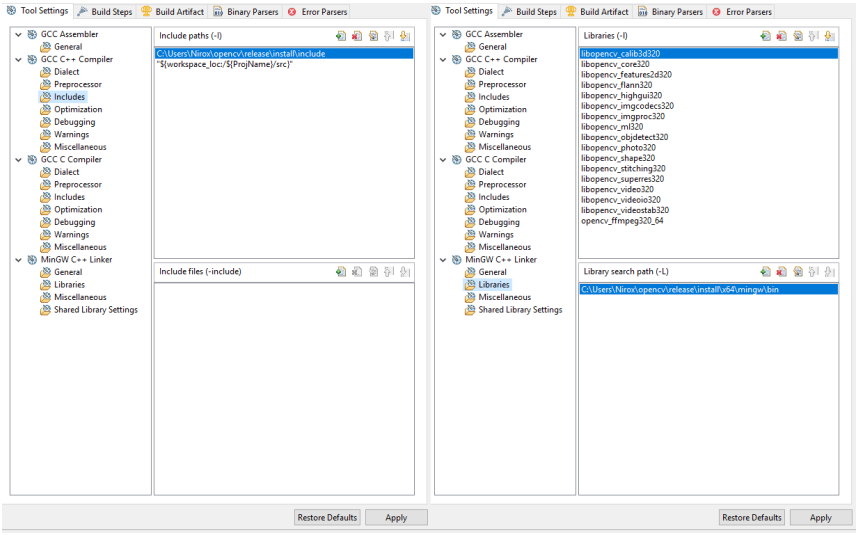


Figura 3.1: Configuración Eclipse

Aplicando este algoritmo a toda la imagen, se pretende encontrar todo el conjunto de tablones de madera de características similares.

Con el objetivo de explicar el algoritmo de una forma detallada, se describen cada una de las funciones utilizadas para el análisis y finalmente la función que realiza el movimiento de la ventana deslizante y hace uso de todas las funciones detalladas.

Inicialización de la imagen

Para el proceso de análisis realizado, la imagen es inicialmente convertida a escalas de grises, ya que solo es utilizada una capa de intensidades y no es necesaria información sobre el color.

Con el objetivo de minimizar las condiciones del entorno de la propia imagen, tales como nivel de luz, sombras, brillo, ruido, etc, se han aplicado un conjunto de técnicas para el tratamiento de la imagen:

- **Normalización del brillo**: se ha realizado una función que calcula la intensidad media de todos los píxeles y posteriormente es eliminada de dichos píxeles. Para el cálculo de la intensidad media se ha hecho uso del histograma de la imagen, ya que facilita el número de píxeles que tienen una determinada intensidad, por lo tanto, la suma de todos los píxeles multiplicado por la intensidad de cada uno, dividido entre el número total de píxeles, nos proporciona el brillo medio. Aplicando esta técnica se consigue una minoración del efecto del brillo y la luz de la imagen en el algoritmo.

```
void normalizeBrightness(cvMat &image){
        /* ... */
        cvcalcHist(&image, 1, 0, cvMat(), hist, 1, &
           histSize, &histRange, true, false);
        float brightness = 0;
        for (int i = 0; i  histSize; i++) {
                brightness += ihist.atfloat(i);
        }
        brightness = (image.cols  image.rows);
        for (int i = 0; i  image.rows; i++) {
                for (int j = 0; j  image.cols; j++) {
                        image.atuchar(i, j)= abs(image.
                           atuchar(i, j) - brightness);
                }
        }
}
```

Código 3.1: Método *normalizeBrightness*

- **Filtro Gaussiano**: se ha aplicado un filtro gaussiano a la imagen, con un determinado sigma, con el objetivo de reducir el ruido existente producido en la propia captura de la imagen. Este ruido produce cambios bruscos en la intensidad de los distintos píxeles de la imagen produciendo un efecto no deseado al aplicar el detector de bordes Canny.

En el algoritmo se pretende encontrar lineas pertenecientes al borde de los tablones de madera, por este motivo se ha decidido utilizar el algoritmo de detección de bordes Canny que nos proporciona una localización mucho mas precisa que otros detectores de bordes. Este hace uso de un conjunto de técnicas:

- **Derivada de la Gaussiana (DroG)**: utilizado para eliminar ruido y detectar los cambios intensos en los píxeles.
- **Cálculo del modulo y dirección del gradiente**: permite obtener la dirección del borde y utilizado para la aplicación de la técnica supresión non-máxima.
- **Supresión non-máxima**: técnica que siguiendo la dirección de los gradientes y el valor del módulo elimina los puntos que no sean máximos locales y permite aminorar el número de falsos positivos en la detección.
- **Histéresis**: consiste en la aplicación de un umbral a partir del cual se definen que píxeles son los considerados con suficiente intensidad para ser detectados como pertenecientes al borde.

El algoritmo de Canny proporciona una imagen que representa de forma binaría los contornos de los tablones, utilizada posteriormente para localizar las 4 líneas que localizan al tablón

de madera.

La imagen proporcionada, en algunos casos, es de un tamaño demasiado grande para su procesamiento de forma rápida en un dispositivo móvil. Por este motivo, se ha decidido realizar una redimensión de la imagen que finalmente va a ser tratada para la localización. Para ello, se ha aplicado una redimensión de un cierto factor que es realizado en la parte final de la función encargada de la inicialización de la imagen, ya que si es realizada con anterioridad se podría perder una gran parte de información de píxeles utilizada por lo anteriores algoritmos y que pueden ser útiles para una mejor localización y conteo de tablones. (Figura 3.2)

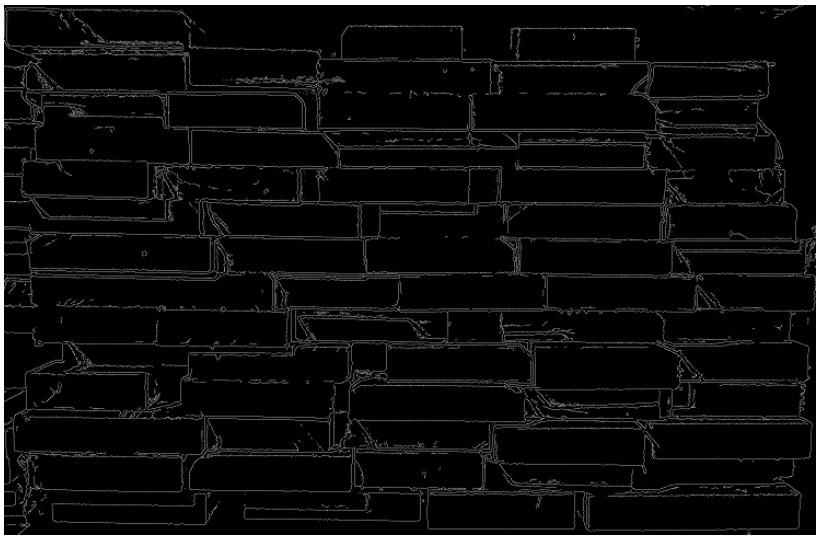


Figura 3.2: Imagen inicializada

```
void imageInitialization(cv::Mat& image, cv::Mat &imageOutput
    , double threshold, double sigma, double factor) {
        threshold *= 255;
        cv::cvtColor(image, imageOutput, CV_BGR2GRAY);
        normalizeBrightness(imageOutput);
        cv::GaussianBlur(imageOutput, imageOutput, cv::Size(2
            * ceil(2*sigma) + 1, 2* ceil(2*sigma) + 1), sigma
           ,sigma,cv::BORDER_REPLICATE);
        cv::Canny(imageOutput, imageOutput, 0.4 *threshold,
           threshold, 3, true);
        cv::resize(imageOutput, imageOutput, cv::Size(), 1 /
           factor, 1 / factor, cv::INTER_CUBIC);
        cv::resize(image, image, cv::Size(), 1 / factor, 1 /
           factor, cv::INTER_CUBIC);
}
```

Código 3.2: Método *imageInitialization*

Análisis de intersección de la ventana

Con el objetivo de agilizar el algoritmo y reducir su tiempo de ejecución, se ha realizado una función encargada de detectar si la ventana actual, intersecta con uno de los rectángulos ya detectados. En el caso de que esta intersección se produzca, la función devuelve una localización donde la ventana actual debe saltar. De esta forma conseguimos evitar el procesamiento de todas las ventanas que produzcan una intersección con algunos de los rectángulos ya detectados.

Para el cálculo del salto, se ha definido una región crítica centrada en el centro de la ventana actual, en la cual, si existe algún rectángulo detectado anteriormente que tenga su píxel central dentro de esta zona, se considera que la ventana actual no podrá detectar ningún otro rectángulo en esta iteración, pues la superficie restante no es suficiente para que ningún otro rectángulo pueda ser localizado.(Figura 3.3)

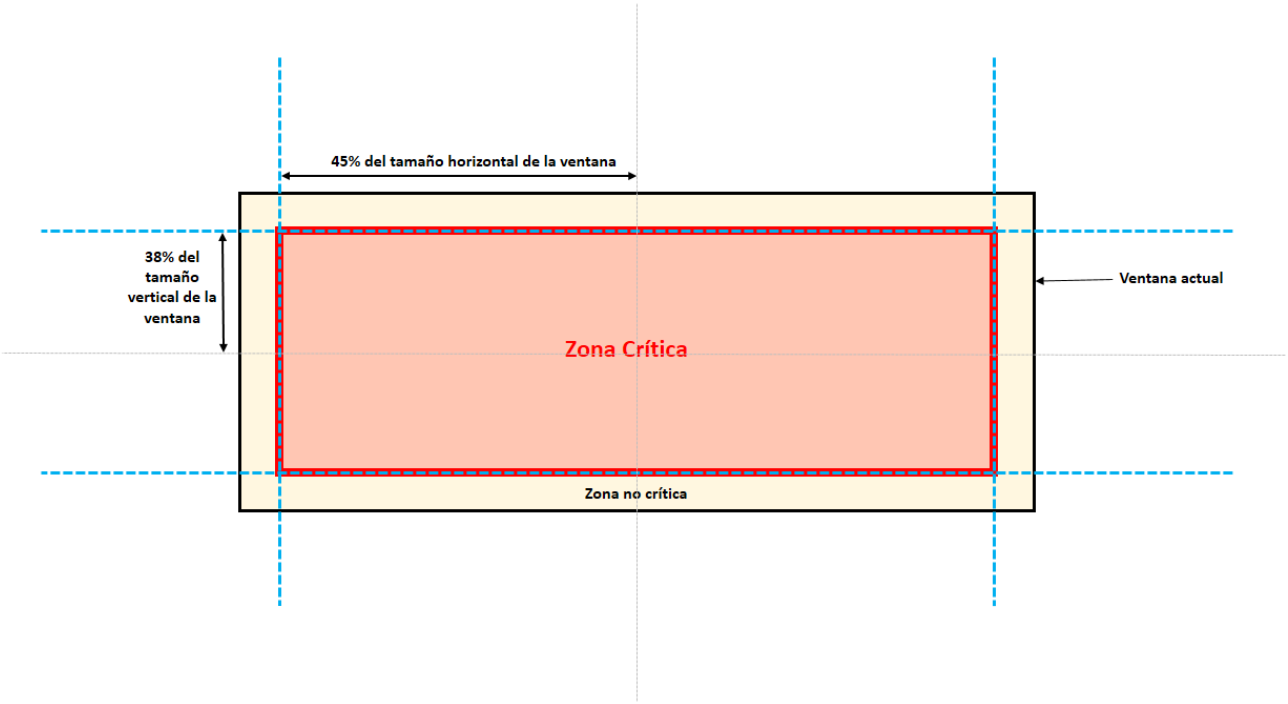


Figura 3.3: Zona crítica de *CheckIntersection*

Si se encuentra uno o varios rectángulos con su centro en esta zona crítica, se selecciona el rectángulo con la linea horizontal inferior más baja, y se devuelve la posición de su línea vertical derecha, pues es utilizada para el posterior salto de la ventana.

```
int checkIntersectionWindow(std::vector<std::array<double,
   12>> rectangles, unsigned rowIndex, unsigned colIndex,
   unsigned lengthH, unsigned lengthV) {
        /* ... */
        for (unsigned i = 0; i < rectangles.size(); i++) {
                if ((std::abs(windowCenter[0] - rectangles[i
                   ][10]) < round(
                   HORIZONTAL_INTERSECT_PERCENT * lengthV))
                   && (std::abs(windowCenter[1] - rectangles[
                   i][11]) < round(VERTICAL_INTERSECT_PERCENT
                    * lengthH))) {
                        currentLine = 2 * (rectangles[i][10]
                           - rectangles[i][8]) + rectangles[i
                           ][10] - rectangles[i][8];
                        if (currentLine > lowestLineValue) {
                                lowestLineValue = currentLine
                                   ;
                                idxLowestLine = i;
                        }
                }
        }
        return idxLowestLine != -1 ? 2 * (rectangles[
           idxLowestLine][11] - rectangles[idxLowestLine][9])
            + rectangles[idxLowestLine][9] : -1;
}
```

Código 3.3: Método *checkIntersectionWindow*

Transformada de Hough

Una vez inicializada la imagen y comprobada la posición de la ventana, se realiza una detección de líneas formadas por los contornos de los tablones mediante la utilización de la transformada de Hough y un conjunto de filtros implementados con el objetivo de rechazar todas las líneas no consideradas pertenecientes al tablón de madera.

Para la realización de la transformada de Hough se ha decidido aceptar solo lineas con una inclinación de -5 y 5 grados respecto a la horizontal y de igual manera para la vertical. Utilizando, por lo tanto, las lineas con grados de 85 a 90 y -5 a 5 respectivamente. También se ha hecho uso de una serie de umbrales que limitarán la intensidad mínima que debe tener una línea para ser considerada válida, estos umbrales son diferentes para las líneas horizontales y verticales, pues dependen del tamaño de la ventana utilizada.

Este primer filtrado permite aminorar el número de líneas consideradas validas y por lo tanto disminuir el tiempo de ejecución del algoritmo.

La implementación de la transformada de Hough ofrecida por OpenCV, no permite filtrar estos dos rangos en una sola ejecución, por lo que se optó por modificar su implementación para que devolviera un parámetro que sería necesario para seguir la implementación realizada en Matlab y se volvió a compilar la librería. Una vez realizada la modificación y compilación, se observó que el algoritmo aumentada en unos 10 segundos su tiempo de ejecución debido a las llamadas de la nueva librería implementada y se intentó realizar una implementación utilizando dos llamadas a la función utilizada para la realización de Hough, obteniendo así un resultado más eficiente, el cuál, ha sido usado en la implementación de la librería.

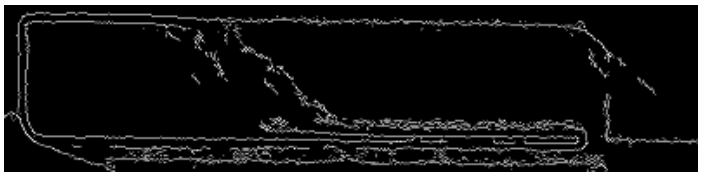
Las líneas obtenidas mediante la transformada, son descritas utilizando una representación polar (ρ, θ). Aunque para el dibujado de los rectángulos, las lineas son obtenidas mediante la ecuación de curvas polares, durante algunas partes de los distintos filtros realizados, se utiliza ρ sin realizar una previa transformación a coordenadas cartesianas. Esto se debe al ángulo utilizado en los filtros, pues al ser ángulos de un intervalo muy pequeño tanto para las horizontales como verticales, el error es irrelevante al no realizar dicha transformación. Dicho error es de $\cos\theta|\theta \in [-5, 5]°$ para las horizontales y de $\sin\theta|\theta \in [85, 95]°$ para las verticales.

Una vez obtenidas las líneas, aunque estas hayan sido filtradas por ángulo e intensidad, la transformada devuelve una gran cantidad de líneas que deben ser descartadas (figura 3.4). Para proceder a la selección, se ha decidido que solo serán devueltas cuatro líneas, dos verticales y dos horizontales. Estas lineas son seleccionadas fijando inicialmente la localizada en la primera posición de la lista de lineas, pues la función de hough devuelve estas líneas ordenadas de mayor intensidad a menor intensidad. Esta primera selección se debe al significado de la intensidad o acumulado de una línea en la transformada de Hough, ya que representa la línea con mayor coincidencia en la imagen binaría.

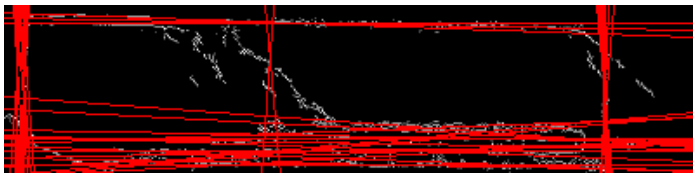
Cuando la línea de mayor intensidad es seleccionada, se siguen recorriendo las demás lineas de su misma orientación, realizando una comprobación más respecto a la línea de mayor intensidad, pues una de las líneas restantes es considerada perteneciente al rectángulo, si la distancia entre la línea de mayor intensidad y la línea estudiada, es mayor que el 50 % del tamaño del lado correspondiente a la ventana actual, es decir, en las horizontales, la segunda horizontal, debe estar a una distancia mayor al 50 % del tamaño vertical de la ventana. Para el cálculo de esta distancia se han definido dos funciones, pues mediante la ecuación de las curvas polares (1) obtiene dos puntos pertenecientes a la recta, localizados a un 25 % de distancia de cada extremo de la intersección de la línea estudiada y la ventana.(Figura 3.5)

$$x\cos\theta + y\sin\theta = \rho$$

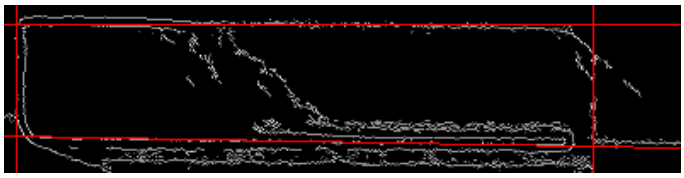
(1)



(a) Ventana sin líneas detectadas



(b) Ventana con líneas detectadas



(c) Ventana con líneas filtradas

Figura 3.4: Líneas detectadas por *houghTransform*

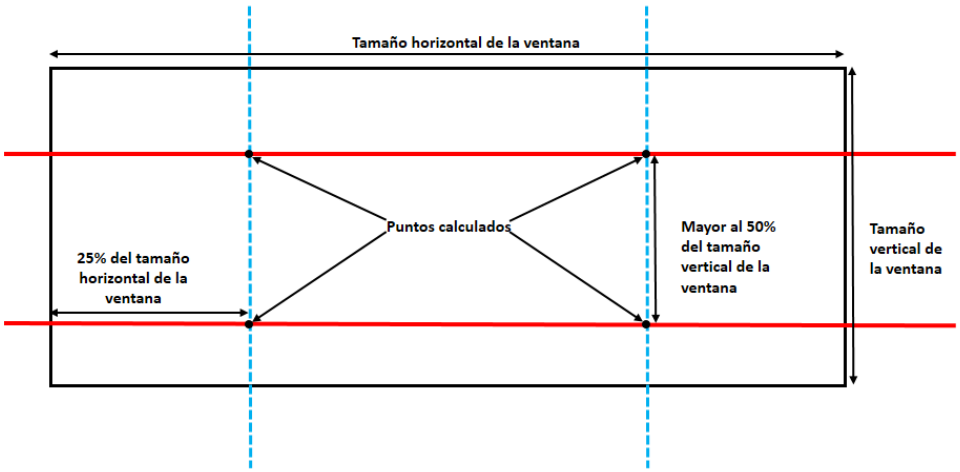


Figura 3.5: Cálculo de puntos para la distancia

Por último, la función devuelve un *booleano* indicando si existen dos lineas horizontales y dos líneas verticales que cumplan las condiciones definidas anteriormente y sus localizaciones polares.

```
bool houghTransform(cv::Mat image, unsigned rowIndex,
   unsigned colIndex,double rectangle[], unsigned thresholdH,
    unsigned thresholdV, unsigned lengthH, unsigned lengthV)
   {
        /* ... */

        cv::HoughLines(image, houghHorizontalLines, RHO_HOUGH
           , THETA_HOUGH, thresholdH, 0, 0, D85_TO_RADIANS,
           D95_TO_RADIANS);
        cv::HoughLines(image, houghVerticalLines, RHO_HOUGH,
           THETA_HOUGH, thresholdV, 0, 0, -D5_TO_RADIANS,
           D5_TO_RADIANS);

        /* ... */

        while ((idxHorizontal < houghHorizontalLines.size()
           || idxVertical < houghVerticalLines.size()) && (
           horizontalLines.size() < 2 || verticalLines.size()
            < 2)) {
                // horizontal
                if((idxHorizontal < houghHorizontalLines.size
                   ()) && horizontalLines.size() < 2){
                     if(horizontalLines.size() != 0){
                             lineHorizontalPoints(
                                coordenatescurrentLine,
                                lengthH,colIndex,
                                houghHorizontalLines[
                                idxHorizontal][0],
                                houghHorizontalLines[
                                idxHorizontal][1]);
                     }
                     if (horizontalLines.size() == 0 || ((
                        std::abs(std::abs(
                        yCoordenatesFirstLineHorizontal
                        [0])-std::abs(
                        coordenatescurrentLine[0])) >
                        THRESH_DISTANCE_V_PERCENT *
                        lengthV) && (std::abs(std::abs(
                        yCoordenatesFirstLineHorizontal
                        [1])-std::abs(
                        coordenatescurrentLine[1])) >
```

```
                        THRESH_DISTANCE_V_PERCENT *
                        lengthV))) {
                            horizontalLines.push_back( {
                                houghHorizontalLines[
                                idxHorizontal][0],
                                houghHorizontalLines[
                                idxHorizontal][1] });
                    }
                }

                // vertical
                /* ... */
        }

        detect = horizontalLines.size() == 2 && verticalLines
            .size() == 2;

        /* ... */

        return detect;
}
```

Código 3.4: Función *houghTransform*

Análisis y dibujado del rectángulo detectado

En esta parte de algoritmo, se analiza la intersección del rectángulo detectado con los otros ya detectados anteriormente, con el objetivo de que este no intersecte, más de un cierto límite definido, a los otros rectángulos.

Este nuevo filtro, consiste en iterar por todos los rectángulos detectados y calcular la distancia desde sus puntos centrales al punto central del nuevo rectángulo. Si la distancia es menor del 80 % del tamaño de la horizotal de la ventana, en el caso de que se esté comparando la distancia horizontalmente, el nuevo rectángulo quedará descartado, pues se considera que intersecta demasiado con algunos de los otros rectángulos detectados anteriormente, devolviendo un *booleano* que indica que no ha sido dibujado ni guardado. La comparación de la distancia entre los centros, no se realiza solo horizontalmente, sino que también se comparan verticalmente

En el caso de que se considere que no intersecta a otro rectángulo, se procederá al cálculo de una serie de datos necesarios para proceder a su dibujo en la imagen de salida y a su guardado.

Para dibujar los rectángulos, se ha creado una nueva función, que introduciendo los datos de dos lineas devuelve el punto de intersección de ambas. Este punto es calculado mediante el punto de corte de dos curvas polares obtenido de desarrollar las siguientes ecuaciones:

$$\{ x \cos\theta_1 + y \sin\theta_1 = \rho_1 x \cos\theta_2 + y \sin\theta_2 = \rho_2$$

Solución:

$$x = \frac{\rho_2 - y \sin\theta_2}{\cos\theta_2}$$

$$y = \frac{\rho_2 \cos\theta_1 - \rho_1 \cos\theta_2}{\sin\theta_2 \cos\theta_1 - \sin\theta_1 \cos\theta_2}$$

Una vez obtenidos los puntos de corte, se dibujan las 4 líneas de punto a punto y para la correcta visualización del rectángulo, se dibuja un punto en la parte central del mismo.

Detector de rectángulos

Esta función es la utilizada para ejecutar todo el algoritmo de detección de tablones de madera y es la ofrecida por la librería para su ejecución. Se encarga de la llamada a las funciones explicadas anteriormente y en el control de los saltos y movimientos de la ventana deslizante.

Inicialmente se realiza la **inicialización de la imagen**, obtienendo una imagen binaria con los bodes detectados y redimensionada. Posteriormente se ejecuta el movimiento de la ventana deslizante de izquierda a derecha y de arriba a abajo.

Una vez obtenida la localización de la ventana, se realiza sobre la imagen binarizada, un recorte que denota los limites de ésta, y se estudia si en la posición donde se encuentra hay intersección con otro rectángulo detectado anteriormente, ejecutando el **análisis de intersección de la ventana**. Si existe intersección se ejecuta un salto de la ventana a la localización proporcionada por el análisis y si no existe se realiza la transformada de Hough de la imagen.

De la realización de la **transformada de Hough**, se obtiene si se han detectado suficientes líneas para la localización. Si este conjunto de líneas no es suficiente se realiza un movimiento más pequeño que al localizar una interseción o un rectángulo, el tamaño de este salto es proporcionada en el algoritmo de manera estática. En el caso de que el conjunto de líneas sea suficiente se realiza un análisis al rectángulo.

Este análisis es realizado en la parte del algoritmo llamada **análisis y dibujado del rectángulo**, y devuelve si el rectángulo tiene una intersección suficiente con otros y si se ha realizado el dibujado. Si se ha podido dibujar, la ventana se sitúa en la parte derecha del rectángulo dibujado menos un margen, pues la línea derecha de este último rectángulo, puede pertenecer a la línea vertical izquierda del siguiente. Si no se ha podido realizar el dibujado, la ventana realiza el mismo movimiento que el realizado cuando no se encuentran suficientes

líneas para componer el rectángulo.

Por último, se hace una comprobación del movimiento de la ventana calculada para la siguiente ejecución, pues si el salto localiza a la ventana en la parte derecha de la imagen y la ventana sale del tamaño total, este movimiento es recalculado para colocar la ventana justo en el límite. Este método también es utilizado cuando la ventana sale de la imagen en la última ejecución vertical.

La ejecución termina cuando se ha ejecutado todas las ventanas cubriendo la imagen completa y devuelve la imagen con los rectángulos dibujados, redimensionada, y el número de rectángulos detectados(Figura 3.6).

Figura 3.6: Detección de rectángulos (67 detectados)

El algoritmo recibe como parámetros la imagen para la que se realiza la detección, una lista vacía donde serán proporcionadas las localizaciones de los rectángulos detectados, el factor de redimensión de la imagen, y los demás parámetros utilizados para la inicialización de la imagen y umbrales de detección, que serán los más influyentes para la detección del algoritmo. Estos últimos parámetros son:

- **Tamaño de la ventana:** es la dimensión usada para la detección y es proporcionada mediante dos parámetros que indican el tamaño vertical y el horizontal. Si la ventana no es de suficiente tamaño, todo rectángulo no intersectado dentro de la misma, no será detectado, y si es demasiado grande, de manera que intersecte a más de un rectángulo,

puede provocar errores en su localización y conteo. Este parámetro será proporcionado por el usuario mediante la realización de una selección de ventana de manera visual.

- **Umbrales de intensidad mínima de líneas:** Utilizado para la detección de líneas de la transformada de Hough, son proporcionados como dos umbrales, uno para la mínima intensidad vertical y otro para la horizontal. Indican la intensidad mínima que debe tener una línea para ser considerada perteneciente al rectángulo.

- **Umbral de Canny:** parámetro utilizado en la inicialización de la imagen, en la detección de bordes usando el algoritmo de Canny. Si este umbral es demasiado alto, la imagen de bordes tendrá mucho ruido y la detección no será correcta, y si es demasiado pequeño la detección de bordes no será suficiente y el algoritmo no detectará la mayoría de los rectángulos

- **Sigma:** parámetro utilizado en la inicialización de la imagen, en el filtro aplicado para la eliminación de ruido. Si este parámetro es muy grande, la imagen resultante después de su filtrado estará muy suavizada o difuminada, lo que provocará, en la detección de bordes realizada posteriormente, una detección insuficiente. Por otro lado, si este parámetro es demasiado pequeño, la imagen resultante tendrá demasiado ruido y Canny detectará demasiados falsos positivos.

  Los tres últimos parámetros y otras constantes, han sido estudiados mediante un algoritmo de fuerza bruta aplicado a un conjunto de imágenes, lo que ha permitido obtener una media de parámetros para la realización de una buena detección. Estos parámetros serán proporcionados de manera estática en la ejecución del algoritmo en Android y su obtención es explicada en el apartado **pruebas**.

3.2. Android

Para el desarrollo de la aplicación Android se ha decidido usar el IDE *Android Studio*, debido al conjunto de herramientas ofrecidas para el desarrollo, fácil instalación de otras librerías, a su herramienta de gestión automatizado de *builds* (*Gradle*) y a su anterior uso en otros proyectos con aplicaciones Android.

El proyecto ha sido realizado para la versión mínima de Android 4.1 (API 16), ya que es la mínima versión necesaria para poder usar las herramientas útiles en el desarrollo de esta aplicación. Esta versión tiene un alto porcentaje de distribución de *smarthphones* que pueden ejecutarla, pues según la información proporcionada por *Android Studio* es de 95,2 %, información que también se ha tenido en cuenta para el desarrollo.

3.2.1. Desarrollo de Actividades

Android divide su desarrollo principalmente en el uso de actividades, que son las encargadas de manejar el funcionamiento principal de la aplicación y de la comunicación con la interfaz

gráfica programada en XML. Para la comunicación entre las distintas actividades y aplicaciones internas se usa un elemento llamado *Intent*, por el cual, mediante una serie de parámetros permite enviar información entre actividades y entre otras aplicaciones del dispositivo.

Para la realización de distintas operaciones que interactúen con las herramientas proporcionadas por el dispositivo, usa un sistema de permisos de aplicación que son inscritas en el *AndroidManifest*, un archivo proporcionado en el proyecto que permite el manejo de los permisos y de otras opciones de la aplicación.

Los permisos usados por la aplicación son:

- **Lectura y escritura en el almacén externo:** permisos necesarios para el guardado de imágenes temporales y para la opción de guardado proporcionada.

Para explicar el desarrollo realizado en Android, se detallan las distintas actividades implementadas y como se han realizado las distintas acciones proporcionadas en cada una de ellas.

Carga de imágenes

Esta actividad proporciona las acciones de carga de imágenes que pueden ser obtenidas desde la galería del dispositivo o capturada desde la propia cámara. Para ello se muestran dos botones.

Cuando el botón de la cámara es pulsado, el evento es capturado por el método *onClick* de la actividad. Al recibir el identificador del botón de la cámara crea un nuevo *Intent* que hace una llamada a esta mediante el uso de la *MediaStore*. Antes de ser lanzado, se crea el archivo temporal donde va a ir almacenada la imagen capturada, creado mediante una clase estática implementada. Una vez el archivo es creado se lanza el *Intent*, lanzando la aplicación de la cámara para realizar la captura.

Al pulsarse el botón de la captura de la galería, se produce un proceso similar al de la cámara, pero lanzando el *Intent* con otro parámetro que permitirá lanzar la galería del dispositivo para la obtención de la imagen.

Una vez el usuario a capturado la imagen o la ha seleccionado de la galería, se ejecuta el método *onActivityResult*, que será el encargado de tratar el archivo obtenido por cualquiera de las dos acciones disponibles en la actividad. Cuando este es lanzado, identifica cual de las acciones fue pulsada, carga la imagen en el dispositivo usando un *bitmap* guardandola en la clase estática *UtilsPhoto*, y por último lanza mediante un *Intent* la actividad de **recorte**.

Recorte

Esta acción de recorte ha sido implementada con el objetivo de que el usuario pueda realizar una detección en solo una parte de la imagen seleccionada. Para ello se ha hecho uso de

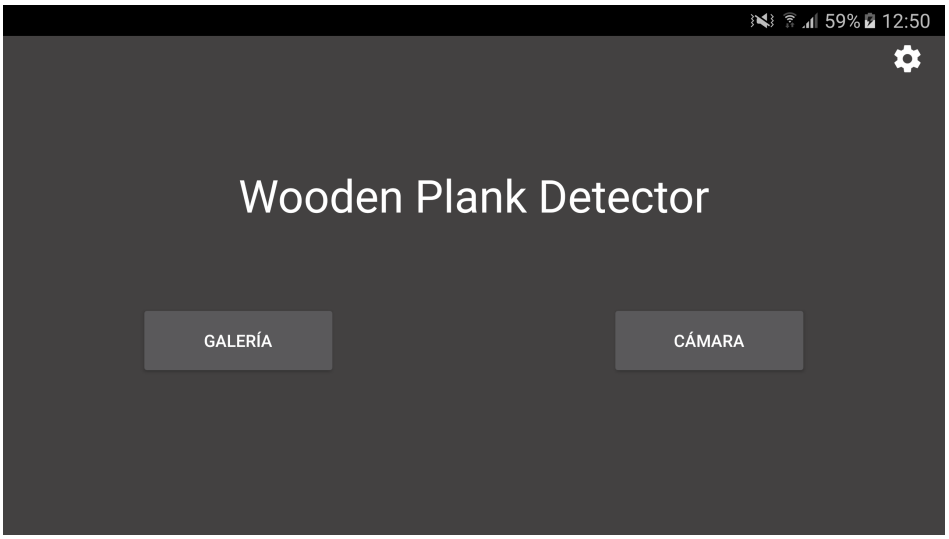


Figura 3.7: Interfaz para la carga de imágenes

una librería llamada *Cropper* (Edmodo) que ofrece esta funcionalidad.

Al inicio de la actividad se muestra la imagen cargada por el usuario y las siguientes acciones:

- **Recortar:** si el usuario pulsa en recortar, el método de captura del evento es lanzado y habilita la interfaz en la que se muestra un rectángulo con el puede realizar el recorte. Si el usuario vuelve a pulsar en recortar, la imagen se guarda en otro *bipmap* diferente al de la imagen cargada, permitiendo la acción de reintentar, mediante la cual, al usuario se le muestra el área de recorte sobre la imagen inicial. Por otro lado, si el usuario pulsa sobre el botón cancelar cuando la selección de recorte está activa, se desactiva la selección de recorte.

- **Aceptar:** si el usuario pulsa sobre aceptar, la imagen que se esté mostrando actualmente en pantalla, es la seleccionada para la detección. Mediante el uso del capturador de eventos del botón se lanza un *Intent* para abrir la siguiente actividad de **selección de ventana**.

- **Cancelar:** si el usuario pulsa sobre el botón cancelar se lanza el capturador de eventos y en el caso de que la selección de recorte no esté activa, se ejecuta el método *onBack-Pressed* que es el encargado, en Android, de volver a la actividad que previamente lanzó a la actividad actual.

Selección de ventana

Mediante esta actividad el usuario puede seleccionar un tamaño de ventana suficiente para la detección.

(a) Interfaz para el recorte de imágenes

(b) Interfaz con el recorte activado

Figura 3.8: Interfaz de recorte

Al inicio de la actividad, se muestra el rectángulo de recorte usado en la actividad anterior. Cuando la selección realizada sea la adecuada, el usuario debe pulsar sobre el botón aceptar, que ejecuta el *Intent* correspondiente para la llamada de la siguiente actividad. El tamaño del rectángulo seleccionado, es el tamaño guardado en la clase *Utilsphoto* para su posterior uso en la detección.

En el caso de que el usuario pulse en cancelar, se ejecuta el método *onBackPressed* que llama a la actividad que lanzó a la actividad actual y por lo tanto, vuelve a la interfaz de recorte.

Figura 3.9: Interfaz para la selección del tamaño de ventana

Detección de tablones

Esta actividad es la encargada de la ejecución del algoritmo de detección de tablones y por lo tanto de la llamada a la librería C++.

Una vez la actividad es lanzada, se realiza la carga de la librería C++ mediante el uso del método *loadLibrary* ofrecido por la clase *System*. Este método lleva como parámetro el nombre de la librería que se quiere cargar, en este caso, *RectangleDetectorLibrary*. El registro de esta librería en el sistema ha sido explicado en el apartado **Instalación de la librería OpenCV y comunicación con librería C++**.

Para poder realizar la detección de rectángulos, se carga la librería para Android ofrecida por OpenCV, además de la librería C++, pues para realizar el tratamiento de la imagen, es necesario convertir la imagen cargada mediante un *bitmap*, al objeto ofrecido por OpenCV llamado *Mat*. Para realizar la carga, es necesario la creación del método *BaseLoaderCallback*. Este método, en el caso de que la librería instalada previamente en el proyecto no tenga la compilación necesaria para la CPU del dispositivo en el que se esté ejecutando, redirecciona al usuario a la tienda android, *PlayStore*, para que pueda descargar una aplicación ofrecida por OpenCV que contiene la librería. Si por lo contrario, la CPU es compatible con algunas de las compilaciones proporcionadas en el proyecto, se realiza la carga de ésta.

Una vez la librería OpenCV es cargada en el dispositivo, se procede a la llamada a la clase *loadRectangleDetector*. Esta clase extiende de la clase proporcionada por Android, *AsyncTask*, pues permite la ejecución de un proceso en una hebra de manera asíncrona, evitando la ejecución en la hebra principal. Si está ejecución se hiciera en la hebra principal, el sistema pararía la aplicación, ya que esta solo puede mantener la hebra principal ejecutando un proceso durante un tiempo determinado por el sistema.

La clase *loadRectangleDetector* implementa los siguientes métodos y funciones necesarios para la ejecución de la hebra:

- **OnPreExecute**: este método es el llamado antes del proceso de ejecución en segundo plano de la hebra. Este es el encargado de la inicialización de un *ProgressDialog* que ofrece una visualización de carga mientras se realiza el proceso de detección.

- **doInBackground**: es la función encargada de la ejecución del proceso en segundo plano. Es donde se realiza la conversión de la imagen cargada en un *Bitmap* a *Mat* y la llamada al método *RectangleDetectorNativeClass.rectangleDetector(...)*, que devuelve la imagen con los rectángulos dibujados y el número de rectángulos detectados. Este método es el encargado de realizar la llamada mediante el *JNI* a la librería C++, cuyo método de creación es explicado en el apartado **Instalación de la librería OpenCV y comunicación con librería C++**. Una vez obtenida la imagen, se realiza la conversión de *Mat* a *Bitmap* y se devuelve el número de rectángulos.

- **onPostExecute**: este método es ejecutado cuando la función *doInBackground* devuelve el número de rectángulos detectados, y muestra en la interfaz, la nueva imagen de los rectángulos dibujados, el número de rectángulos y detiene el *ProgessDialog*.

Una vez el proceso de detección ha finalizado el usuario tiene disponible las siguientes opciones:

- **Guardar:** permite al usuario guardar la imagen en su dispositivo, esta acción es recogida por el capturador de eventos, desde el cual, se realiza el guardado mediante el uso de un método implementado en la clase estática *Utilsphoto*. Si la imagen es guardada correctamente, muestra un mensaje indicándole al usuario que la imagen a sido guardada correctamente.

- **Compartir:** permite al usuario compartir la foto mediante algunas de las aplicaciones de comunicación del dispositivo. El mensaje enviado contiene un texto con el número de tablones detectados y la imagen. Para realizar la acción se hace uso de un *Intent* que utiliza el *flag Intent.ACTION_SEND* para enviar los datos.

- **Cancelar:** ejecuta el método *onBackPressed* que ha sido modificado para eliminar los archivos temporales necesarios en la ejecución actual de la aplicación y para lanzar mediante un *Intent* a la primera actividad de la aplicación dedicada a la carga de imágenes.

Otras configuraciones

En Android, las actividades cumplen con un ciclo de vida que le permite al sistema operativo el control de la aplicación ante los distintos eventos ocurridos en el sistema. En la figura 3.11 se puede observar el ciclo de vida de las actividades y su ejecución ante los distintos estados de la aplicación.

Figura 3.10: Interfaz para la detección de tablones

En la aplicación, se han implementado un conjunto de métodos comunes en todas las actividades que no han sido explicados en los apartados anteriores. Estos métodos son:

- **onCreate:** este método es ejecutado al inicio de cada actividad. Es donde se han configurado los distintos elementos de la interfaz y, con el objetivo de controlar algunas de las excepciones que pueden ser lanzadas por la aplicación, se ha sustituido el manejador de excepciones por defecto mediante el uso de dos clases implementadas. La clase *ApplicationIntance*, en la que se guarda una instancia de la aplicación para su posterior uso en la clase *ExceptionHandler*. Ésta es utilizada en el caso de que la aplicación sea cerrada por el sistema, debido al lanzamiento de una excepción no controlada, pues hace que vuelva a ser lanzada mediante el uso de un *PendingIntent* y *AlarmManager*.

- **onDestroy:** este método es ejecutado justo antes de la destrucción de la actividad. Es donde se ha realizado el borrado de los archivos temporales guardados para el uso de la aplicación, evitando así que estos permanezcan en el dispositivo si la actividad es destruida de forma inesperada.

Por otra parte, para que la aplicación esté disponible en varios idiomas, se han utilizado distintos archivos XML que contienen las cadenas de caracteres utilizadas. Si el idioma del dispositivo es distinto al español, la aplicación es mostrada en inglés y caso contrario, en español. También se han realizado distintas configuraciones para los distintos tipos de pantallas de los dispositivos. Estas configuraciones son realizadas en los archivos *dimens*, que al igual que los archivos de idioma, son elegidos por el propio sistema según la configuración del dispositivo y su localización en el proyecto, pues Android utiliza una estructura de archivos predefinida para estas configuraciones.

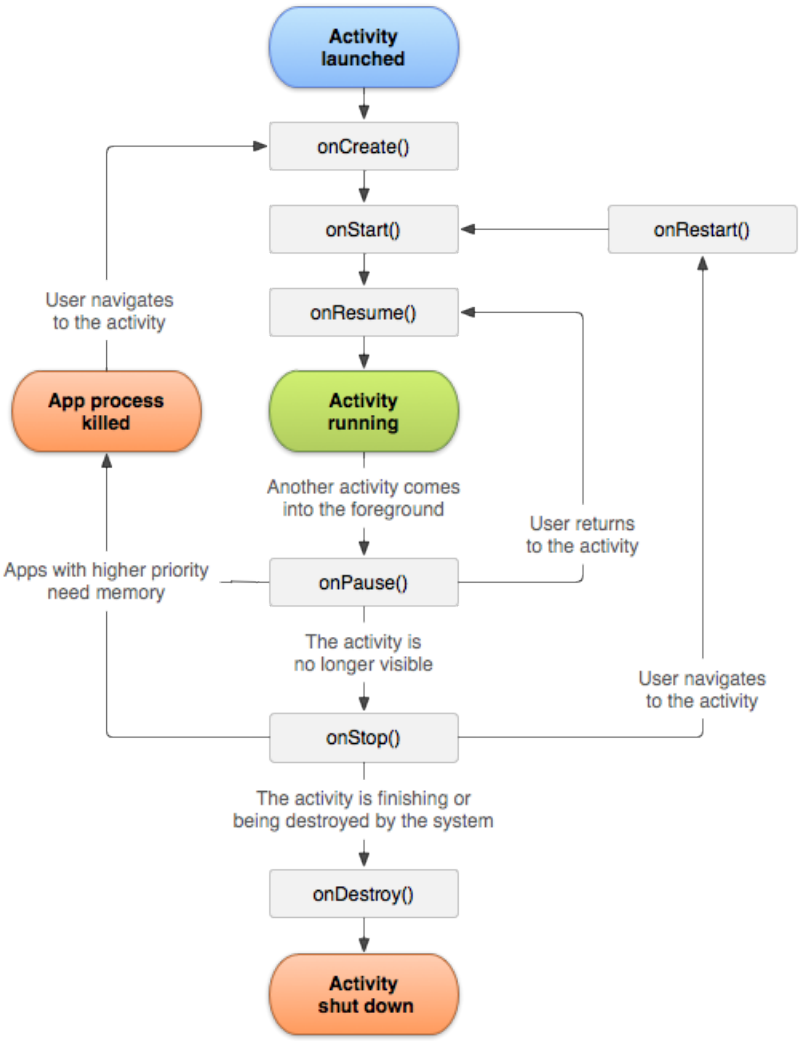


Figura 3.11: Ciclo de vida de actividad Android [1]

3.2.2. Instalación de la librería OpenCV y comunicación con librería C++

Una vez el proyecto a sido creado con la configuración inicial ofrecida por Android Studio, se ha descargado la compilación ofrecida por OpenCV para Android y se ha importado el SDK Java usando las dependencias del proyecto.

Para realizar el enlace con la librería C++, se ha habilitado el uso del NDK en el proyecto introduciendo *android.useDeprecatedNdk= true* en el archivo *gradle.properties*. Después se ha realizado la creación de la clase que realiza la comunicación entre la aplicación Android, mediante el uso del *framework* JNI (*Java Native Interface*). Esta clase se ha llamado *RectangleDetectorNativeClass* y es donde se ha realizado la implementación del método que enlaza con la librería usando una clase C++ generada desde la consola mediante la ejecución del comando: *javah -d jni -classpath ../../build/intermediates/classes/debug*

[1] https://developer.android.com/guide/components/activities.html

com.tfg.woodenplankdetector.Utils.RectangleDetectorNativeClass, siendo el último parámetro del comando, la ruta del paquete donde se encuentra la clase java en el proyecto.

Por último se ha configurado la localización de la librería, el nombre utilizado para cargarla, configuraciones de variables necesarias para la compilación y localización de las librerías C++ previamente compiladas por OpenCV mediante el uso de tres archivos que especifican estas configuraciones *Application.mk, Android.mk y build.gradle*.

4. Pruebas

Se han realizado un conjunto de pruebas de cada unas de las funciones implementadas y se ha comprobado su funcionamiento de manera visual, mediante un conjunto de imágenes. Algunas de las imágenes generadas han sido mostradas en apartados anteriores. En estas pruebas se ha tenido en cuenta el tiempo de ejecución y la exactitud de la localización de tablones, mejorando en algunos casos el algoritmo para aminorar su tiempo de ejecución y localización

Para la realización de las pruebas en Android, se ha aplicado un plan de pruebas para la comprobación del funcionamiento de todas las actividades implementadas, mediante la realización de un conjunto de combinaciones con las acciones de cada actividad.

Por otro lado, con el objetivo de realizar un estudio de parámetros, se han realizado pruebas mediante la ejecución de un algoritmo de fuerza bruta. Este consiste en comprobar todas las combinaciones posibles entre un conjunto de rangos para cada uno de los parámetros influyentes en la detección y localización de tablones. Con este algoritmo, se ha analizado el conjunto de imágenes disponibles, guardando aquellas imágenes en las que se han detectado un determinado número de tablones, correspondiente al número de tablones existentes en la imagen contado de manera manual y aplicando una tolerancia de error, que varía según el estado de la imagen.

Los parámetros estudiados con el algoritmo y los rangos de estudio son:

- **Umbrales de intensidad mínima de líneas:** se ha aplicado unos rangos de $[50, 130]$ con un paso de 10 para la horizontal y $[15, 45]$ con un paso de 5 para la vertical. Se ha observado que depende del tamaño de la ventana seleccionada y de la detección de bordes realizada, pues es el tamaño mínimo de la intensidad de cada línea detectada con la transformada de Hough. Por lo tanto, mientras más ruido tiene la imagen de bordes, mayor debe ser este parámetro y cuanto más pequeña sea la ventana, menor debe ser.
- **Umbral de Canny**: se ha aplicado un rango de $[0{,}045, 0{,}09]$ con un paso de 0.005. Depende del ruido, brillo e intensidad de los contornos de los tablones.
- **Sigma:** se ha aplicado el rango de $[2, 6]$ con un paso de 0.5. Depende principalmente del ruido de la imagen.

Estos rangos han sido acotados progresivamente mediante el algoritmo, siendo los mencionados anteriormente el conjunto de rango correspondiente al último estudio realizado.

Una vez generadas todas las imágenes, se ha realizado un análisis visual para seleccionar aquellas con una mejor detección, obteniendo así un conjunto de parámetros acotados que han sido los utilizados en la llamada a la librería C++ en Android. Con estos parámetros se ha calculado un valor medio de sigma, umbral de Canny y una recta de regresión, con la cual, a partir del tamaño de la ventana se calculan unos valores próximos a los óptimos de los umbrales de intensidad mínima de líneas.

Rectas de regresión:

$$Umbral\ horizontal = 53{,}47 + 0{,}030 * horizontalSize$$

$$Umbral\ Vertical = 11{,}22 + 0{,}065 * verticalSize$$

5. Conclusiones

Los algoritmos de visión por computador tienen aplicaciones en distintos campos, son utilizados en industrias para la automatización de procesos industriales, en la medicina para herramientas de diagnostico automáticos, herramientas de apoyo a la cirugía, telemedicina, etc.

Generalmente, las zonas donde se aplican los algoritmos de visión por computador, suelen ser entornos donde se controlan los niveles de luz, el dispositivo que realiza la captura y varios parámetros que afectan al análisis, permitiendo realizar un mejor ajuste de los parámetros utilizados en los algoritmos aplicados en estos entornos. Este es el mayor problema que se ha intentado mejorar en el algoritmo de detección de tablones, ya que aunque se ha realizado un estudio de parámetros detallado, explicado en el apartado anterior, no se ha podido ajustar para realizar una correcta detección para distintas condiciones del entorno, pues los parámetros del algoritmo son muy sensibles al ruido en la imagen y a los niveles de brillo y contraste.

Para que el usuario de la aplicación pueda realizar una buena detección mediante el uso del algoritmo, se ha decidido implementar una pantalla donde pueda cambiar los parámetros utilizados por el algoritmo, dependiendo del entorno donde vaya a realizar la detección frecuentemente. También se ha añadido un botón que permite volver a los parámetros iniciales de la aplicación, pues son los parámetros obtenidos después del análisis realizado.

5.1. Posibles mejoras

Con el objetivo de mejorar el algoritmo de detección de tablones realizado se han pensado nuevas ideas para mejorar la implementación del mismo:

- **Machine Learning:** realizar la implementación de un módulo de aprendizaje que permita al algoritmo modificar los parámetros utilizados para la detección según una serie de características de la imagen de entrada. El algoritmo se debería someter a aprendizaje mediante un conjunto voluminoso de imágenes de distintas características, de las cuales, se tendrían localizados el centro de cada rectángulo así como su tamaño, permitiendo al algoritmo realizar el aprendizaje para ajustar lo máximo posible los parámetros utilizados a la detección realizada de manera manual con el conjunto de imágenes. Esto permitiría que los parámetros utilizados en la detección fueran dinámicos y mejoraría considerablemente el resultado del algoritmo.

- **Uso del color:** en el algoritmo actual, para realizar la detección no se utiliza la información proporcionada por el color de los tablones y de las demás partes de la imagen, si no que se basa en el tratamiento de los bordes localizados mediante Canny. Para mejorar la localización de los tablones y evitar la posible confusión en la detección de un hueco en el panel de tablones, se podría realizar una mejora en el algoritmo mediante el uso del color de la imagen, de manera que se detectaría el color de los tablones de madera para los que se están realizando la detección y se ajustaría el algoritmo para ignorar las detecciones que no correspondan, aproximadamente, al color de estos tablones.

Bibliografía

[1] Android developer. Referencia de Android.
https://developer.android.com/, visitado por última vez el 24/05/17.

[2] Documentación C++. Referencia de C++.
http://www.cplusplus.com/reference/, visitado por última vez el 05/04/17.

[3] Documentación OpenCV. Referencia de OpenCV.
http://docs.opencv.org/, visitado por última vez el 14/06/17.

[4] Documentación Stack Overflow. Referencia de Android y C++.
https://stackoverflow.com/, visitado por última vez el 19/05/17.

[5] YouTube - OpenCV, NDK y Android Studio. Referencia de Android NDK.
https://www.youtube.com/watch?v=0q3oiCfSgbo, visitado por última vez el 14/05/17.

[6] González Jiménez, Javier. Apuntes de la asignatura Visión por Computador.
https://informatica.cv.uma.es, visitado por última vez el 29/02/17.

[6] Apuntes de la asignatura Visión por Computador. Referencia de C++.
https://informatica.cv.uma.es, visitado por última vez el 29/02/17.

[7] Cropper library. Referencia de Android.
https://github.com/edmodo/cropper, visitado por última vez el 15/05/17.

[8] UML Web Site. Referencia de UML.
http://www.uml.org/, visitado por última vez el 26/06/17.

[9] Manual de LaTex. Referencia de Latex.
https://es.wikibooks.org/wiki/Manual_de_LaTeX, visitado por última vez el 16/05/17.

[10] MagicDraw. Referencia de MagicDraw.
https://www.nomagic.com/products/magicdraw, visitado por última vez el 12/03/17.

Anexos

A. Anexo I: Manual de usuario

Wooden Plank Detector

Manual de Usuario

Índice general

Introducción

Wooden Plank Detector es una aplicación para dispositivos móviles desarrollada con el objetivo de detectar, localizar y realizar un conteo de los tablones de madera perteneciente a una imagen.

La calidad de la detección depende de las características de la imagen dada como entrada en la aplicación ya que el algoritmo es sensible al ruido producido en la captura de la imagen, brillo, color de los tablones, diferencia entre los tablones, fondo de la imagen, etc

Para mejorar la detección se proporcionan una serie de parámetros proporcionados por defecto obtenidos a partir de un conjunto de pruebas con varias imágenes, con la opción de modificación por parte del usuario según las características del entorno donde la foto ha sido realizada y del dispositivo.

Configuración

En este apartado se especifican algunos de los requerimientos necesarios para el correcto funcionamiento de la aplicación.

2.1. Requisitos del sistema

Para el correcto funcionamiento de la aplicación es necesario un dispositivo con un sistema operativo Android, con una versión mínima de Android 4.1 (API 16), cámara accesible desde la aplicación y la aceptación del permiso de almacén para varias funcionalidades implementadas.

2.2. Instalación de la aplicación

La aplicación no está disponible en ningún repositorio de aplicaciones android, por lo tanto para ejecutarlo es necesario obtener el APK o bien su ejecución mediante el uso del código fuente.

2.2.1. Ejecución desde Android

Una vez la aplicación es instalada en el dispositivo aparecerá el ejecutable de la aplicación con el nombre *Wooden Plank Detector* y el logo.

Funcionalidades de la aplicación

La aplicación dispone de un conjunto de pantallas o actividades que serán explicadas a continuación junto con las funcionalidades disponibles en cada una.

Carga de imágenes

Esta actividad proporciona las acciones de carga de imágenes que pueden ser obtenidas desde la galería del dispositivo o capturada desde la propia cámara. Para ello se muestran dos botones.

- **Galería:** permite el lanzamiento de la galería del dispositivo para cargar una imagen desde ésta.
- **Galería:** permite el lanzamiento de la cámara del dispositivo para realizar la captura de la imagen del análisis.

Por otro lado, también proporciona la opción de realizar un **cambio en los parámetros** utilizados para el análisis de la imagen. Esta opción puede ser ejecutada con el botón de opciones disponible en la esquina superior de la pantalla.(Figura 3.1)

Una vez el usuario a capturado la imagen o la ha seleccionado de la galería, se lanza la actividad de **recorte**.

Recorte

Esta pantalla muestra la funcionalidad de recorte, que permite al usuario realizar una detección en solo una parte de la imagen seleccionada (Figura 3.2).

Al inicio de la misma se mostrará la imagen cargada por el usuario y las siguientes acciones:

- **Recortar:** al pulsa en recortar, se habilita la interfaz en la que se muestra un rectángulo con el puede realizar el recorte. Si el usuario vuelve a pulsar en recortar, la imagen obtenido es la utilizada para el análisis y el botón cambia su funcionalidad a reintentar, mediante la cual, se muestra el área de recorte sobre la imagen inicial. Por otro lado, si

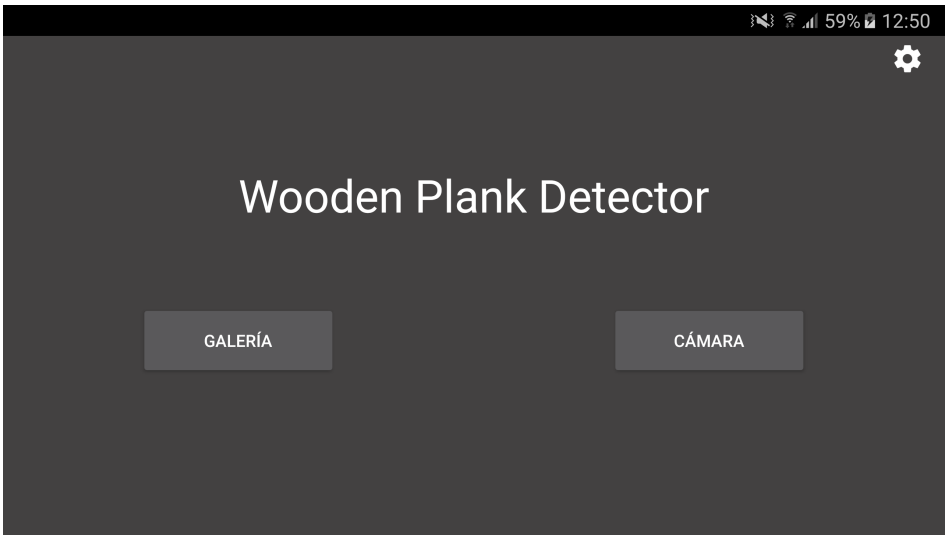


Figura 3.1: Interfaz para la carga de imágenes

se pulsa sobre el botón cancelar cuando la selección de recorte está activa, se desactiva la selección.

- **Aceptar:** al pulsar sobre aceptar, la imagen que se esté mostrando actualmente en pantalla, es la seleccionada para la detección y se lanza la actividad de **selección de ventana**.
- **Cancelar:**al pulsar sobre el botón cancelar, en el caso de que la selección de recorte no esté activa, vuelve a la actividad de **recorte**. Esta opción puede ser ejecutada desde el botón atrás del dispositivo.

Selección de ventana

Mediante esta actividad se puede seleccionar un tamaño de ventana suficiente para la detección, el cual, debe de ser de un tamaño aproximado al de todos los tablones de la imagen.

Al inicio de la actividad, se muestra el rectángulo de recorte usado en la actividad anterior. Cuando la selección realizada sea la adecuada, el usuario debe pulsar sobre el botón aceptar, que lanza la siguiente actividad de **detección de tablones**.(Figura 3.3)

En el caso de que el usuario pulse en cancelar, vuelve a la actividad de **recorte**, disponible desde el botón atrás del dispositivo.

Detección de tablones

Esta actividad es la encargada de la ejecución del algoritmo de detección de tablones.

Al inicio de la actividad, en el caso de que la CPU del dispositivo en el que se esté ejecutando no sea de los modelos instalados por defecto, se redirecciona al usuario a la tienda

(a) Interfaz para el recorte de imágenes

(b) Interfaz con el recorte activado

Figura 3.2: Interfaz de recorte

android, *PlayStore*, para que pueda descargar una aplicación ofrecida por OpenCV que contiene la librería.

Una vez instalado todo lo necesario, se realiza la detección de los tablones utilizando los parámetros proporcionados en la actividad de **carga de imágenes**. Mientras la detección está siendo realizada se muestra un proceso de carga.

Cuando el proceso de detección ha finalizado (Figura 3.4) el usuario tiene disponible las siguientes opciones:

- **Guardar:** permite guardar la imagen en el dispositivo y si es guardada correctamente muestra un mensaje que lo indica.
- **Compartir:** permite compartir la imagen mediante algunas de las aplicaciones de comu-

Figura 3.3: Interfaz para la selección del tamaño de ventana

nicación del dispositivo. El mensaje enviado contiene un texto con el número de tablones detectados y la imagen.

- **Cancelar:** lanza primera actividad de la aplicación dedicada a la **carga de imágenes**, esta opción está disponible desde el botón atrás del dispositivo.

Figura 3.4: Interfaz para la detección de tablones